Introduction to Darshan: What to do when you aren't sure what to do

ATPESC 2020

Phil Carns
Mathematics and Computer Science Division
Argonne National Laboratory

July 31, 2020

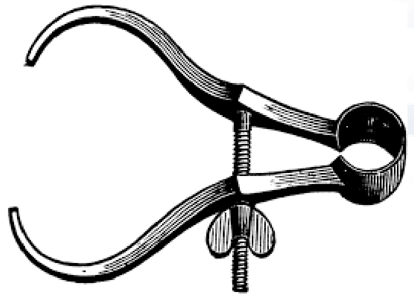
Understanding I/O problems in your application

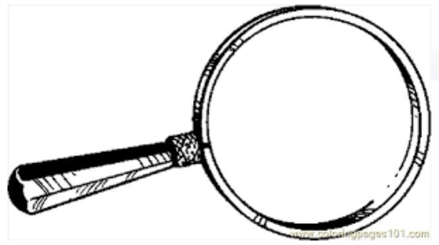
Example questions:

- How much of your run time is spent reading and writing files?
- Does it get better, worse, or the same as you scale up?
- Does it get better, worse, or the same across platforms?
- How should you prioritize I/O tuning to get the most bang for your buck?

We recommend using a tool called **Darshan** as a starting point.

This presentation is a introduction; we'll see more detailed Darshan examples later today.





What is Darshan?

Darshan is a scalable HPC I/O characterization tool. It captures a concise picture of application I/O behavior with minimal overhead.

- Widely available
  - Deployed at most large supercomputing sites
  - Including ALCF, OLCF, and NERSC systems used for ATPESC training
- Easy to use
  - No changes to code or development process
  - Negligible performance impact: just “leave it on”
- Produces a *summary* of I/O activity for every job
  - This is a great starting point for understanding your application’s data usage
  - Includes histograms, timers, counters, etc.

How does Darshan work?

Darshan is primarily intended for MPI applications.* It inserts lightweight instrumentation when your program is compiled and executed.

- Records statistics about file accesses
  - Statistics are stored in bounded, compact memory at each rank
- Aggregates statistics when the application exits
  - Collect, filter, compress and write a single summary file for the job
- Provides command line tools to inspect and interpret statistics
  - Usually start by generating a summary PDF that plots metrics of interest

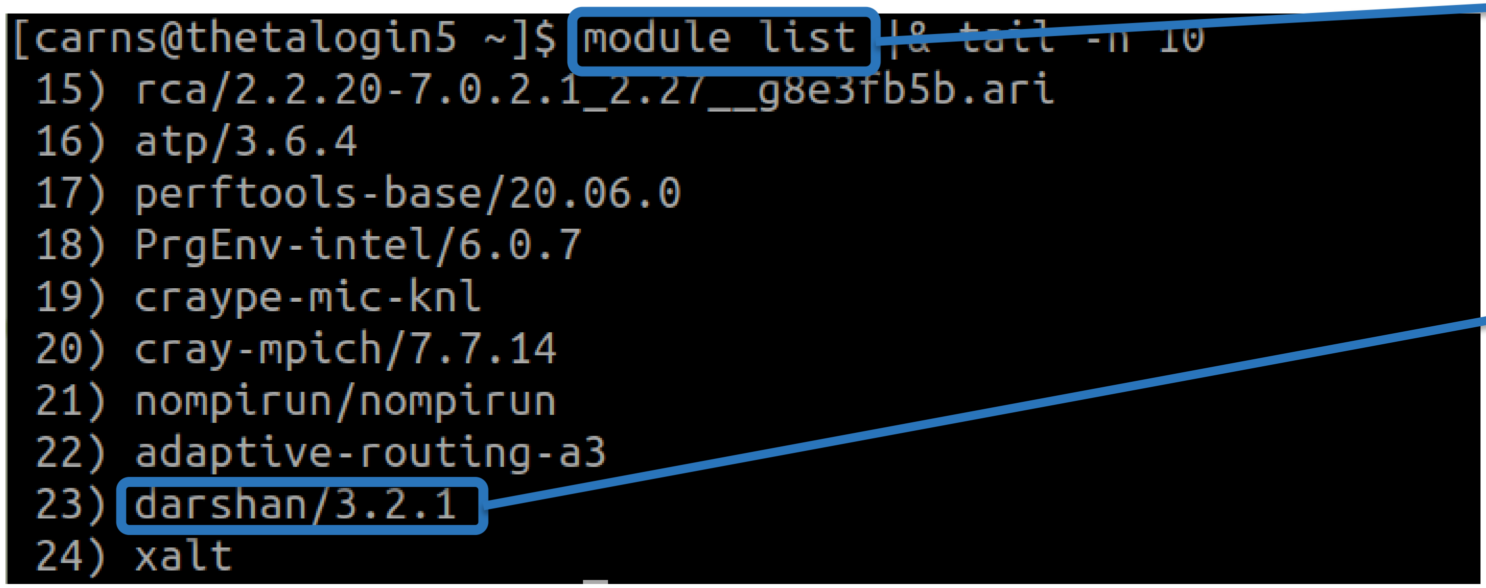
* You can also instrument non-MPI applications; we'll cover this in the afternoon session.

Using Darshan

- We’ll use Theta as an example in the following slides.
- The hands on exercises also include examples that are set up for use on Theta.
  - https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on#darshan
- Other systems are very similar, though. The most likely differences are:
  - Location of log files (where to find data after your job completes)
  - Analysis utility availability (usually easiest to just copy logs to your workstation to analyze)
  - Loading the Darshan module (if it’s not already there by default)
- We’ll briefly cover differences on other DOE systems after the Theta example

Using Darshan on Theta: make sure the software is loaded

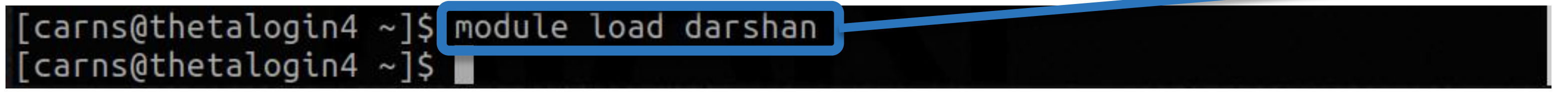
These steps are similar on other platforms; check your site documentation!



```
[carns@thetalogin5 ~]$ module list |& tail -n 10
 15) rca/2.2.20-7.0.2.1_2.27__g8e3fb5b.ari
 16) atp/3.6.4
 17) perftools-base/20.06.0
 18) PrgEnv-intel/6.0.7
 19) craype-mic-knl
 20) cray-mpich/7.7.14
 21) nompirun/nompirun
 22) adaptive-routing-a3
 23) darshan/3.2.1
 24) xalt
```

Use "**module list**" to see a list of software loaded in your environment.

Darshan is probably already loaded by default.



```
[carns@thetalogin4 ~]$ module load darshan
[carns@thetalogin4 ~]$
```

If not, just run "**module load darshan**" to get it.

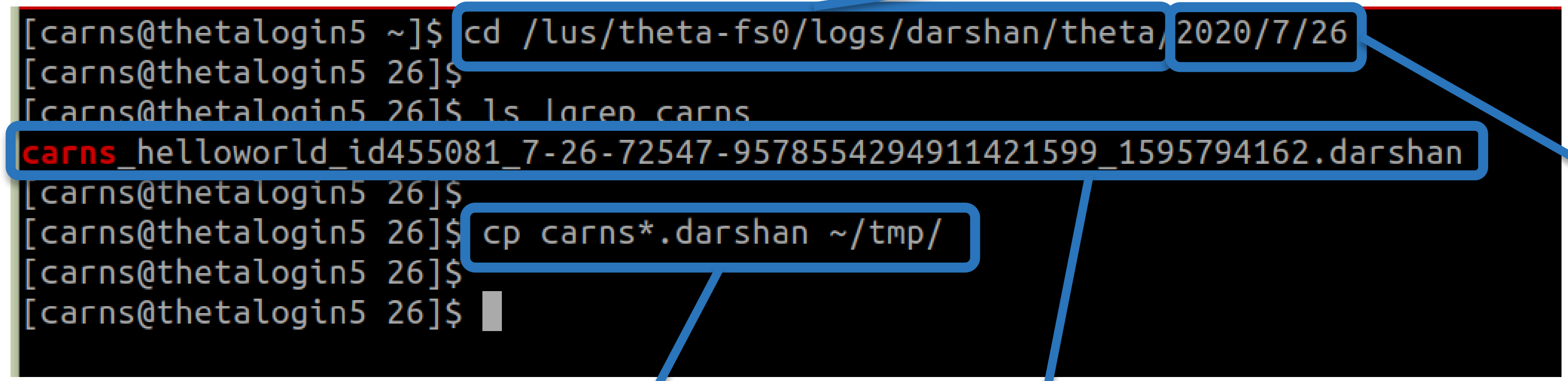
Using Darshan on Theta: instrument your code

Compile and run your application!

That's all there is to it; Darshan does the rest.*

* Well, almost. There is one caveat: in the default Darshan configuration, your application must call MPI_Initialize() and MPI_Finalize() to generate a log.

Using Darshan on Theta: find your log file



```
[carns@thetalogin5 ~]$ cd /lus/theta-fs0/logs/darshan/theta/2020/7/26
[carns@thetalogin5 26]$
[carns@thetalogin5 26]$ ls |grep carns
carns_helloworld_id455081_7-26-72547-9578554294911421599_1595794162.darshan
[carns@thetalogin5 26]$
[carns@thetalogin5 26]$ cp carns*.darshan ~/tmp/
[carns@thetalogin5 26]$
[carns@thetalogin5 26]$
```

All Darshan logs are placed in a central location. This is the path on Theta. **Check your site documentation!**

Go to subdirectory for the year / month / day your job executed.

Be aware of time zone (or just check adjacent days)! Theta, for example, uses the GMT time zone and will roll over to the next day at 7pm local time.

File name includes your username, binary name, and job ID.

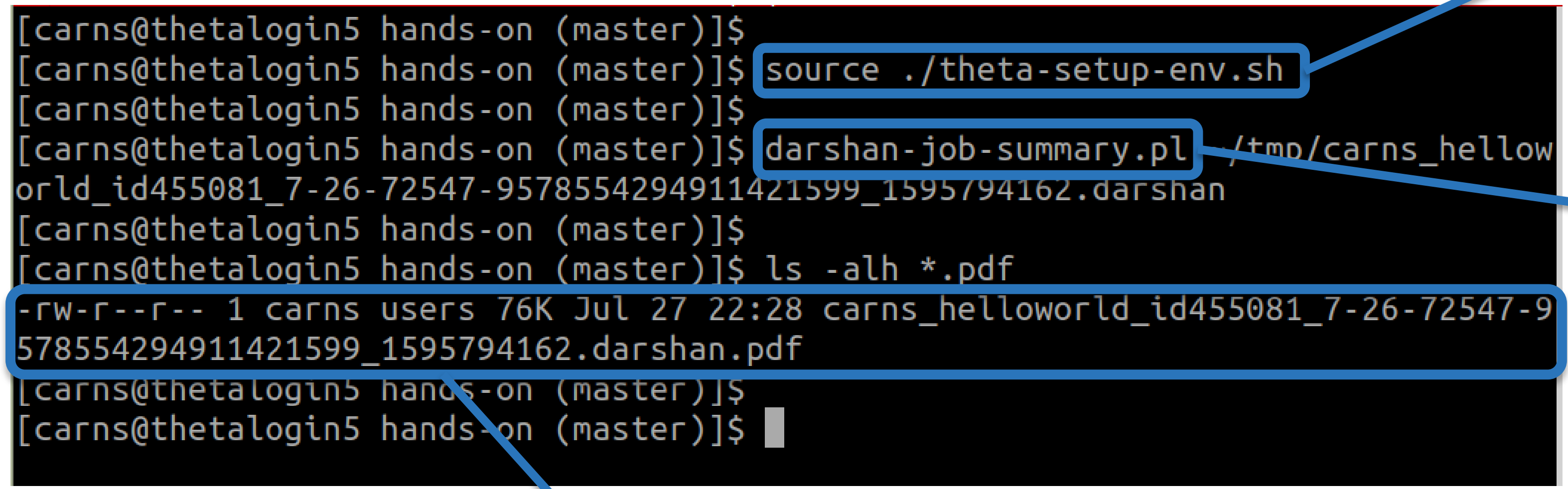
Find the one that you want, and copy it somewhere to analyze.

Hands on exercises: https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

Using Darshan on Theta: generate summary

At this point you could scp the log to another system to analyze (the logs are portable). This example shows how to generate a summary using tools on Theta.

The atpesc-io hands-on exercise repository includes a script to configure your environment with the tools needed for Darshan analysis.



```
[carns@thetalogin5 hands-on (master)]$
[carns@thetalogin5 hands-on (master)]$ source ./theta-setup-env.sh
[carns@thetalogin5 hands-on (master)]$
[carns@thetalogin5 hands-on (master)]$ darshan-job-summary.pl ~/tmp/carns_hellow
orld_id455081_7-26-72547-9578554294911421599_1595794162.darshan
[carns@thetalogin5 hands-on (master)]$
[carns@thetalogin5 hands-on (master)]$ ls -alh *.pdf
-rw-r--r-- 1 carns users 76K Jul 27 22:28 carns_helloworld_id455081_7-26-72547-9
578554294911421599_1595794162.darshan.pdf
[carns@thetalogin5 hands-on (master)]$
[carns@thetalogin5 hands-on (master)]$
```

Use darshan-job-summary.pl to process your log.

It produces a PDF file that (by default) has the same name as your original log, plus a .pdf extension.

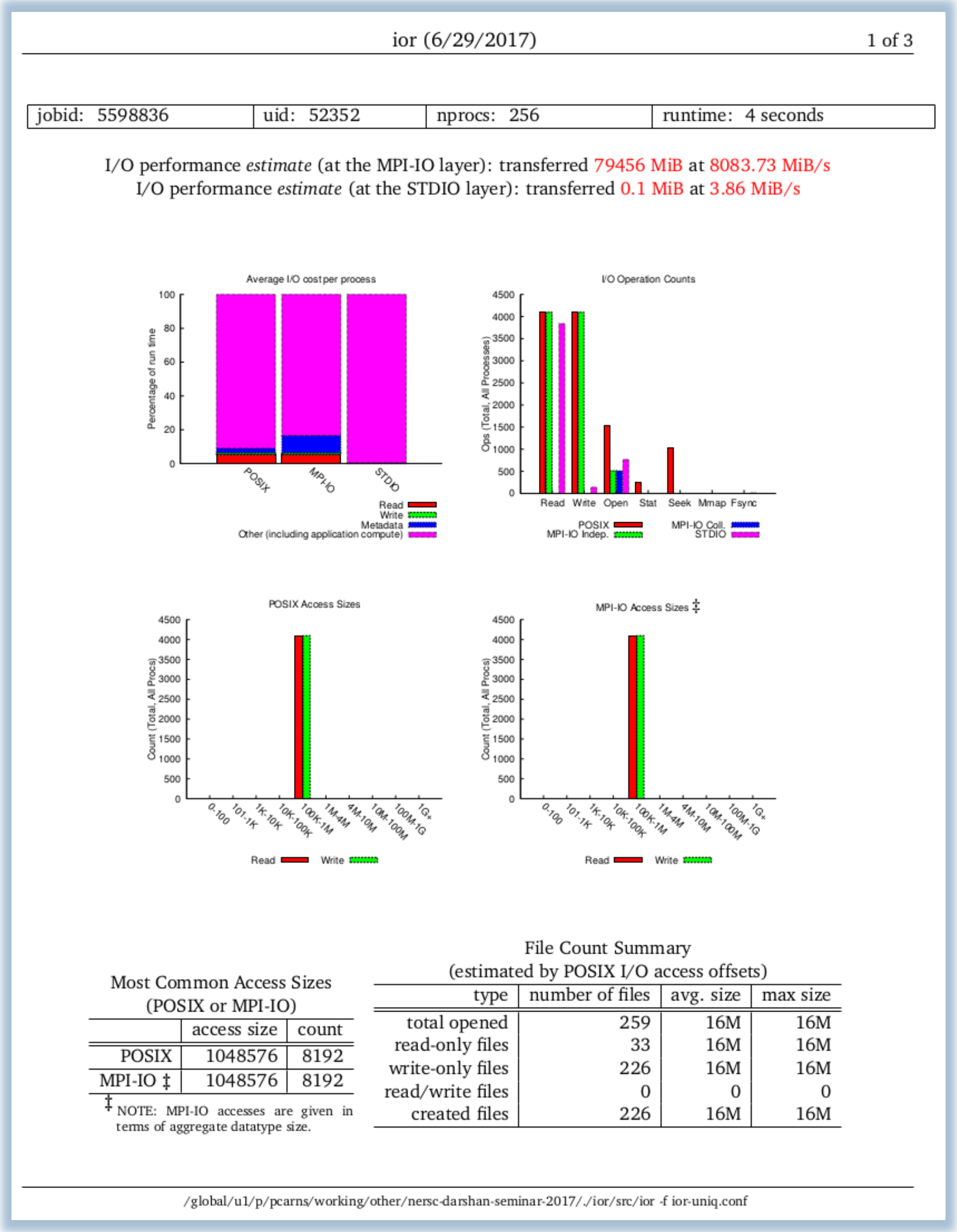
What about other systems?

- Cori:
  - How to enable: automatic
  - Log directory: /global/cscratch1/sd/darshanlogs/
- Summit:
  - How to enable: automatic
  - Log directory: /gpfs/alpine/darshan/summit
- Ascent:
  - How to enable: “module load darshan”
  - Log directory: /gpfs/wolf/darshan/ascent

On each of these systems, you can use darshan-parser to inspect logs in text format, but you will need to copy the logs to another system to generate the pdf summary. Install the “darshan-util” package from Spack or the darshan-util portion of the Darshan source from:

https://www.mcs.anl.gov/research/projects/darshan

Job analysis example



ior (6/29/2017) 1 of 3

jobid: 5598836	uid: 52352	nprocs: 256	runtime: 4 seconds

I/O performance *estimate* (at the MPI-IO layer): transferred 79456 MiB at 8083.73 MiB/s
I/O performance *estimate* (at the STDIO layer): transferred 0.1 MiB at 3.86 MiB/s

Most Common Access Sizes
(POSIX or MPI-IO)

	access size	count
POSIX	1048576	8192
MPI-IO ‡	1048576	8192

‡ NOTE: MPI-IO accesses are given in terms of aggregate datatype size.

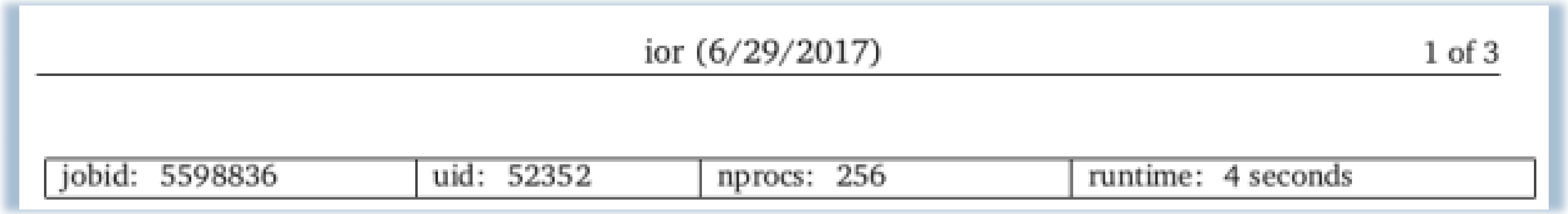
File Count Summary
(estimated by POSIX I/O access offsets)

type	number of files	avg. size	max size
total opened	259	16M	16M
read-only files	33	16M	16M
write-only files	226	16M	16M
read/write files	0	0	0
created files	226	16M	16M

/global/u1/p/pcarns/working/other/nersc-darshan-seminar-2017/./ior/src/ior -f ior-uniq.conf

- The summary PDF contains a few pages of graphs and charts.
- The first page looks like this.
- We'll highlight some key sections in the next slides.

Job analysis example

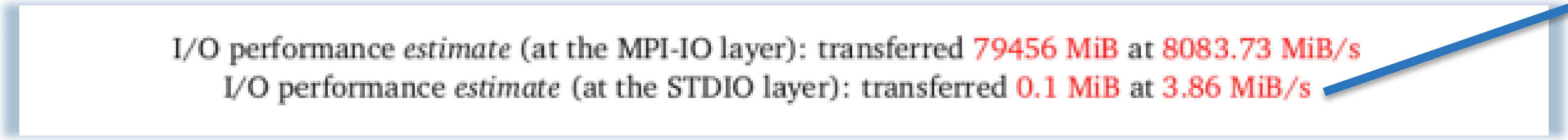


ior (6/29/2017) 1 of 3

jobid: 5598836	uid: 52352	nprocs: 256	runtime: 4 seconds

The header shows basic information about your job:

- Executable name and date
- Job ID, User ID, number of MPI processes, total execution time



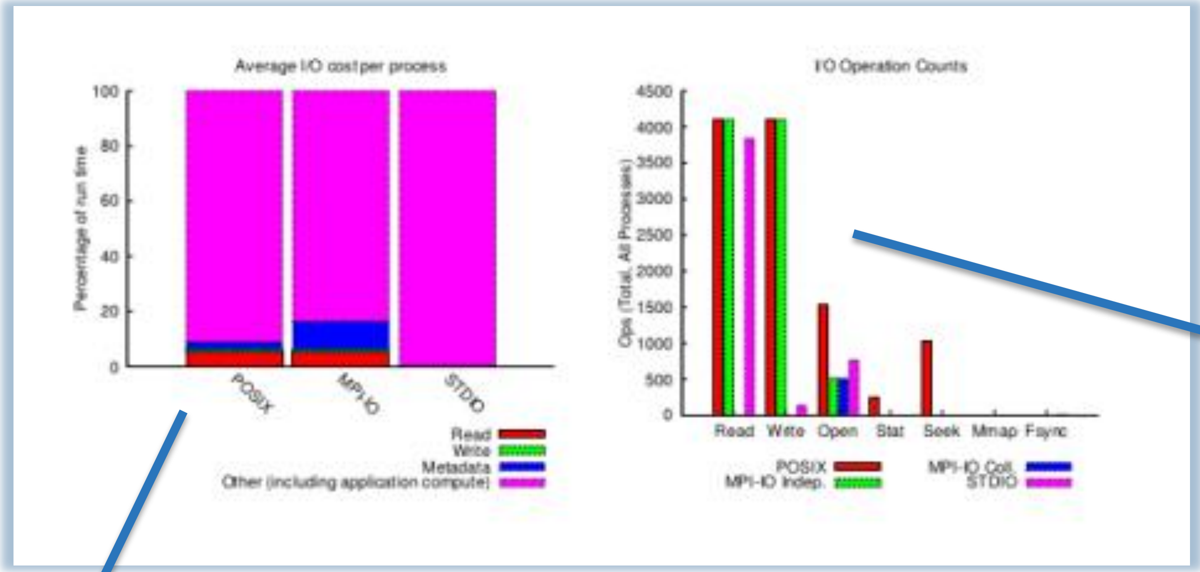
I/O performance *estimate* (at the MPI-IO layer): transferred 79456 MiB at 8083.73 MiB/s
I/O performance *estimate* (at the STDIO layer): transferred 0.1 MiB at 3.86 MiB/s

NOTE: STDIO performance appears low, but that's because it didn't transfer enough data to sustain throughput. In this case this I/O was access to a configuration file.

The next lines show an estimate of your I/O performance. It might display one or more of:

- MPI-IO performance (we'll learn more about this later)
- POSIX performance (open/close/read/write)
- STDIO performance (fopen/fclose/fread/fwrite)

Job analysis example



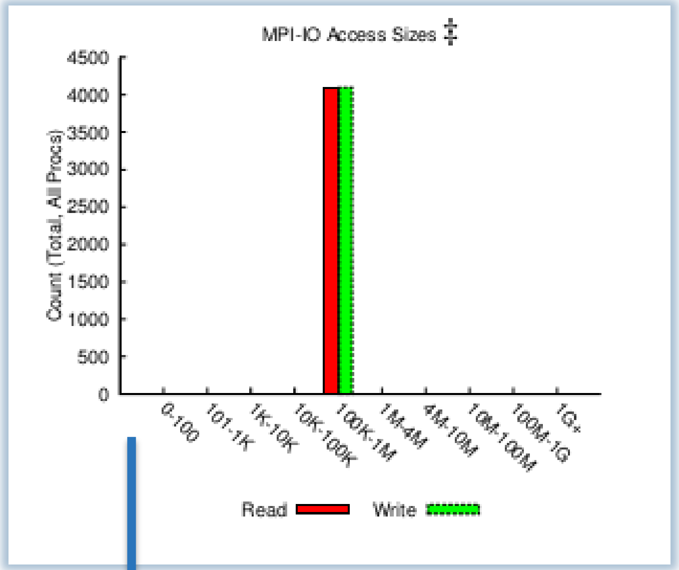
The second graph shows how many times various I/O functions were called.

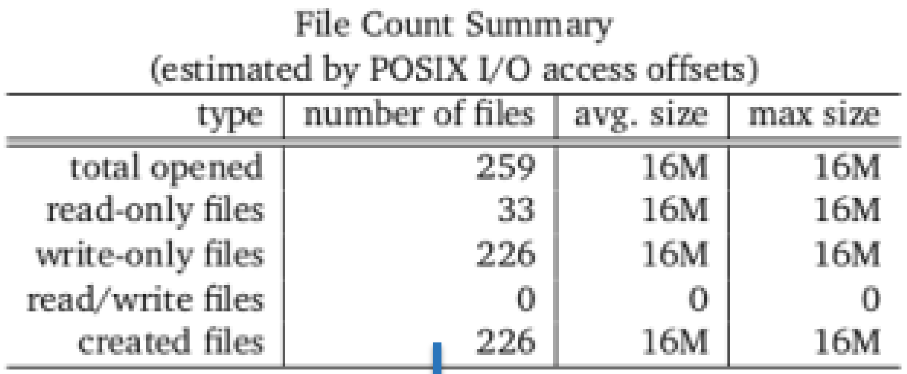
Too many opens or stats could be a warning sign.

The first graph shows the percentage of execution time that was spent performing I/O.

If the percentage is low, then maybe I/O shouldn't be your top priority for optimization?

Job analysis example





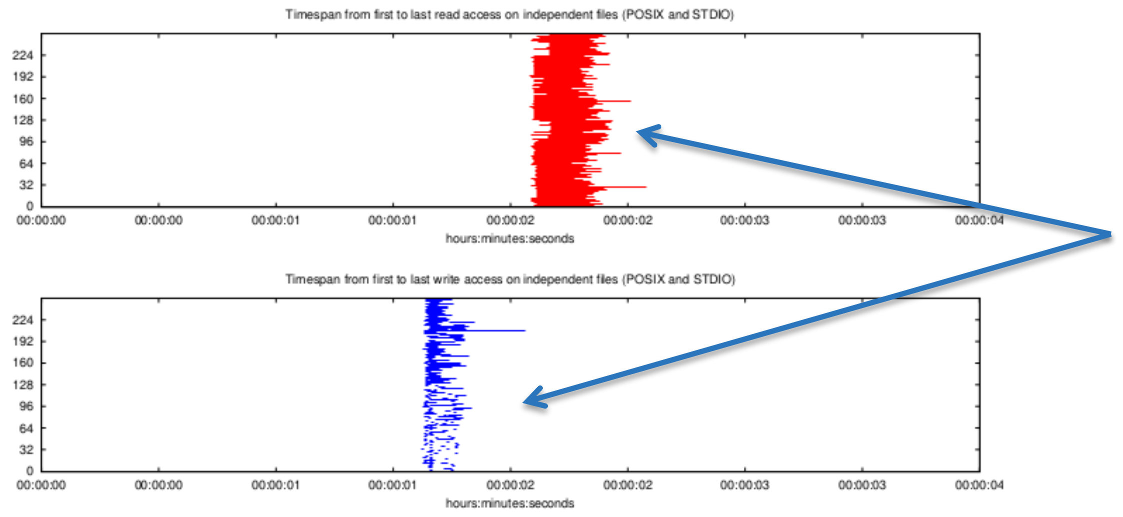
File Count Summary (estimated by POSIX I/O access offsets)			
type	number of files	avg. size	max size
total opened	259	16M	16M
read-only files	33	16M	16M
write-only files	226	16M	16M
read/write files	0	0	0
created files	226	16M	16M

A histogram indicates the distribution of access sizes.

Recall from introduction: if you see many small reads or writes (big spikes on the left hand side), then you are probably not taking advantages of the file system's strongest assets.

A table indicates file counts and sizes for a few different categories of files opened by your application.

Job analysis example



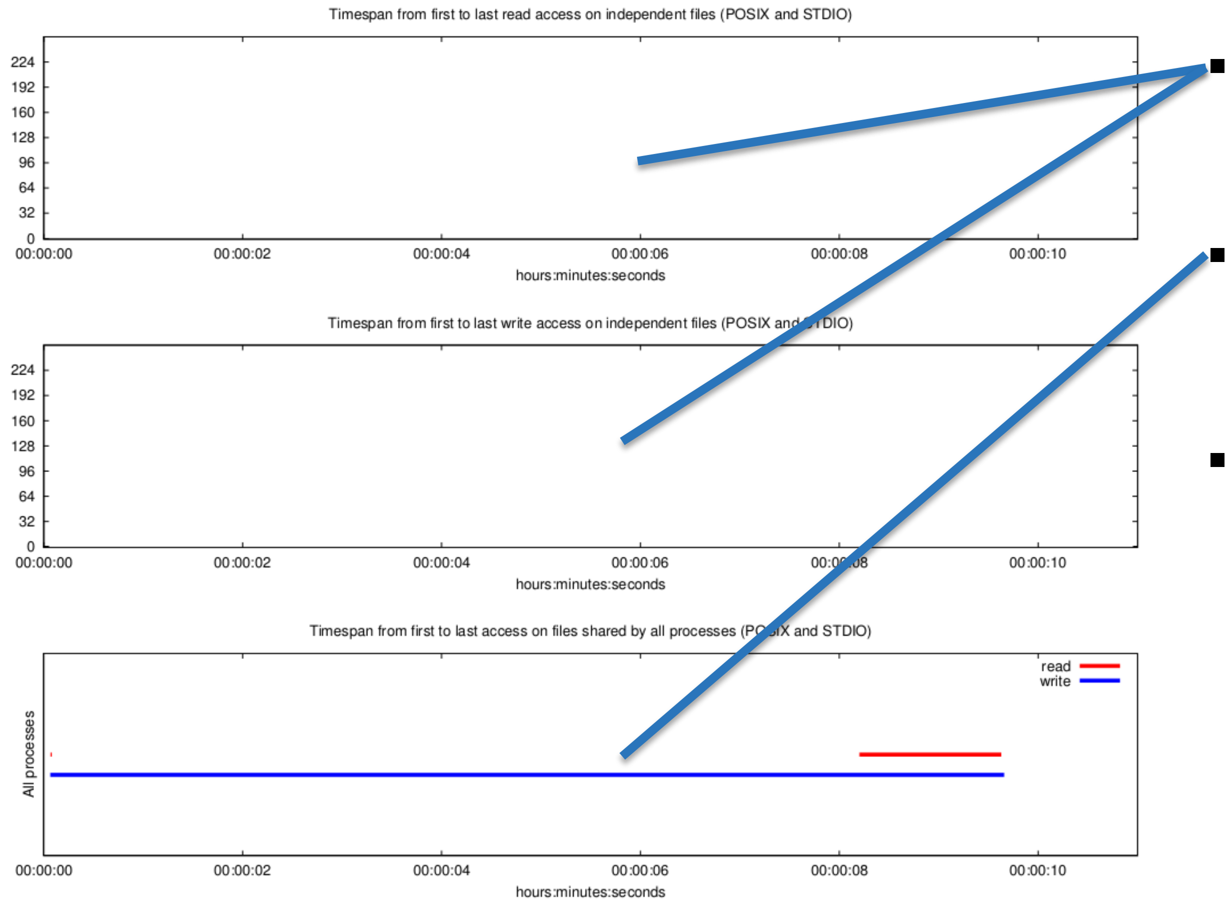
A “timespan” graph on the 2nd page shows the periods of time when the application was reading or writing.

Remember to contact your site’s support team for help! The Darshan summary can be a good discussion starter if you aren’t sure how to proceed with performance tuning or problem solving.

There are additional graphs in the PDF file not shown here. You can also dump all data from the log in human-readable text format using “darshan-parser”.

Darshan: quick tips and tricks

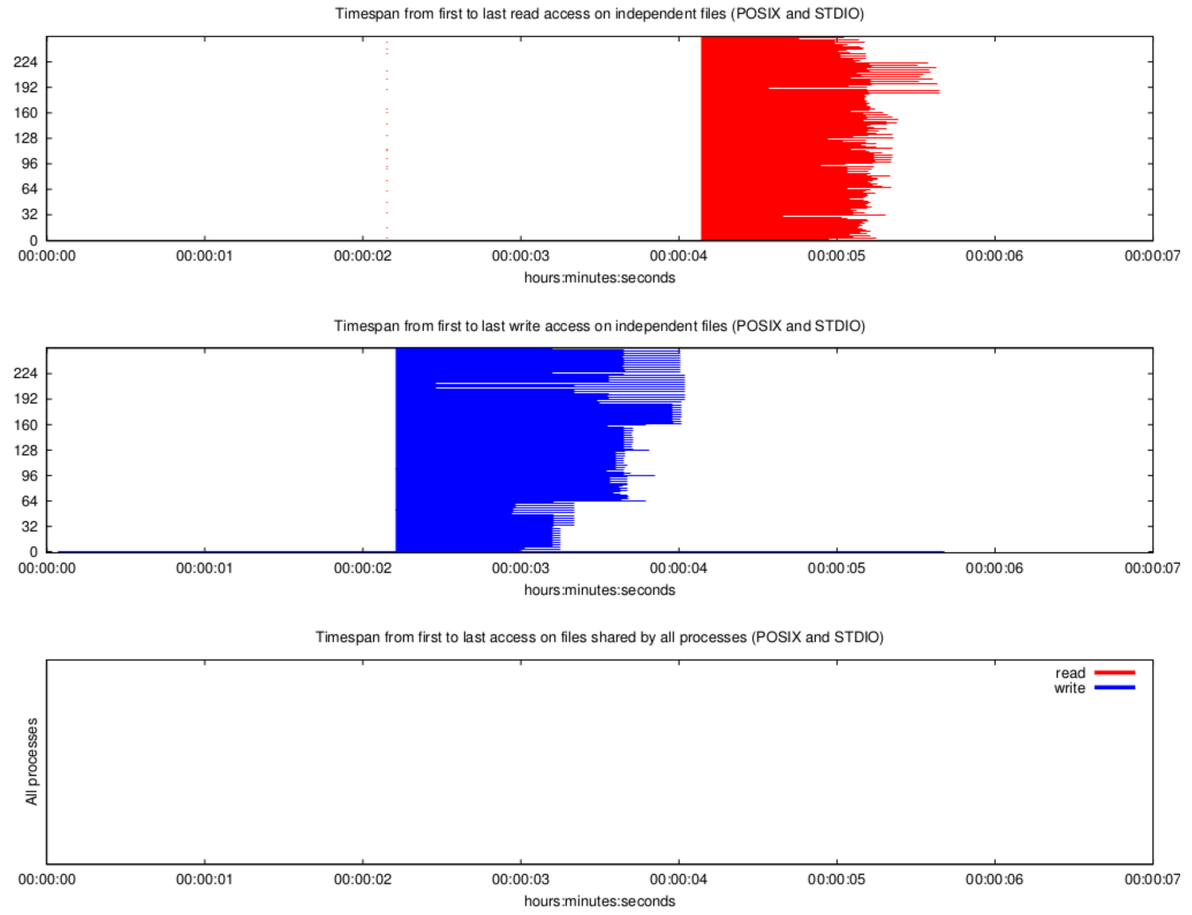
What if you are doing shared-file IO?

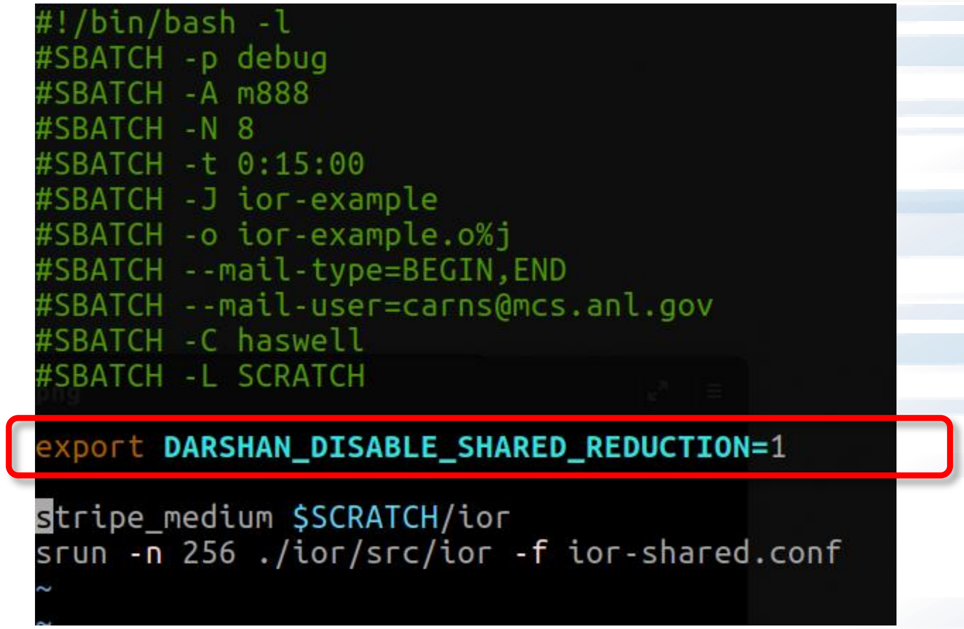


- Your timeline might look like this.
- No per-process information available because the data was aggregated by Darshan to save space/overhead.
- It is shown as just one line for "all processes" in bottom graph.

- Does this matter? The level of detail that you need depends on what you want to learn about your application.

What if you are doing shared-file IO?





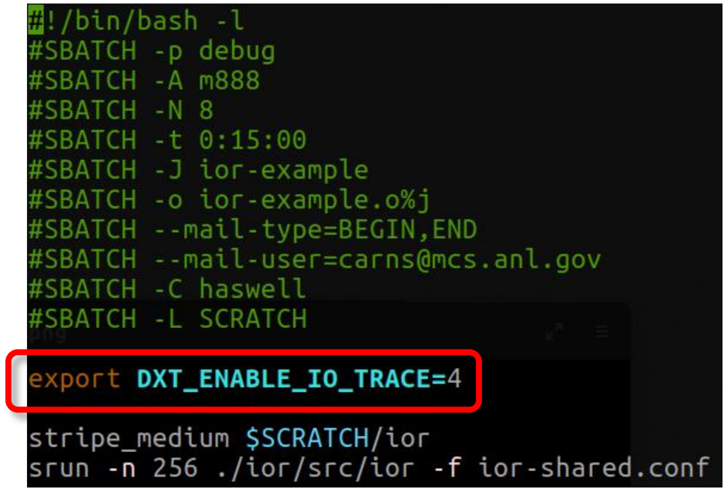
```
#!/bin/bash -l
#SBATCH -p debug
#SBATCH -A m888
#SBATCH -N 8
#SBATCH -t 0:15:00
#SBATCH -J ior-example
#SBATCH -o ior-example.o%j
#SBATCH --mail-type=BEGIN,END
#SBATCH --mail-user=carns@mcs.anl.gov
#SBATCH -C haswell
#SBATCH -L SCRATCH

export DARSHAN_DISABLE_SHARED_REDUCTION=1

stripe_medium $SCRATCH/ior
srun -n 256 ./ior/src/ior -f ior-shared.conf
~
```

- You can optionally set an environment variable in your job, DARSHAN_DISABLE_SHARED_REDUCTION, that tells Darshan *not* to summarize shared file access.
- Every rank will report it's own timespans.
- This increases overhead and log size, but can be very helpful in some cases.

Hands on exercises: https://xgitlab.cels.anl.gov/ATPESC-IO/hands-on

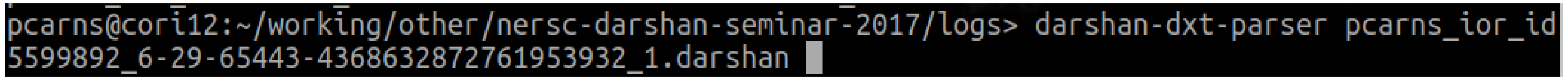
Detailed trace data

- What if that *still* isn't enough detail? You can also capture a full trace with timestamp, offset, and size of every I/O operation on every rank.
- Set the DXT_ENABLE_IO_TRACE environment variable in your job to enable this feature.
- This causes additional overhead and larger files, but captures precise access data.

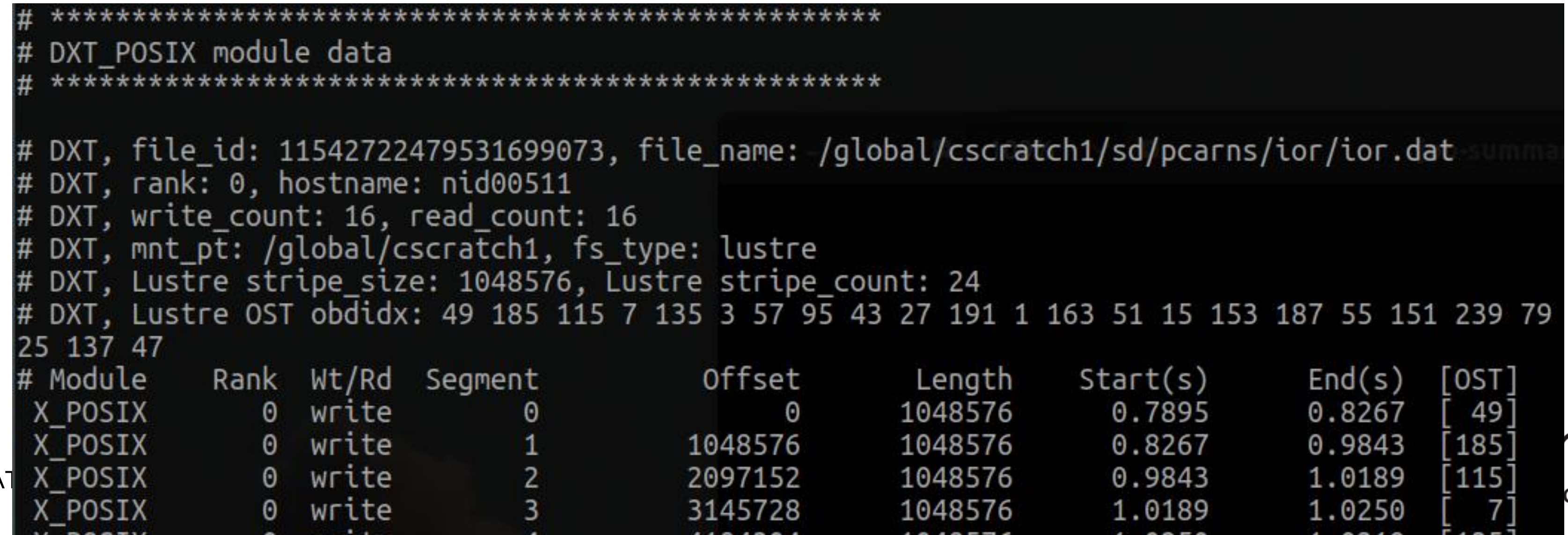


```
#!/bin/bash -l
#SBATCH -p debug
#SBATCH -A m888
#SBATCH -N 8
#SBATCH -t 0:15:00
#SBATCH -J ior-example
#SBATCH -o ior-example.o%j
#SBATCH --mail-type=BEGIN,END
#SBATCH --mail-user=carns@mcs.anl.gov
#SBATCH -C haswell
#SBATCH -L SCRATCH

export DXT_ENABLE_IO_TRACE=4

stripe_medium $SCRATCH/ior
srun -n 256 ./ior/src/ior -f ior-shared.conf
```

You can dump the trace information with "darshan-dxt-parser."

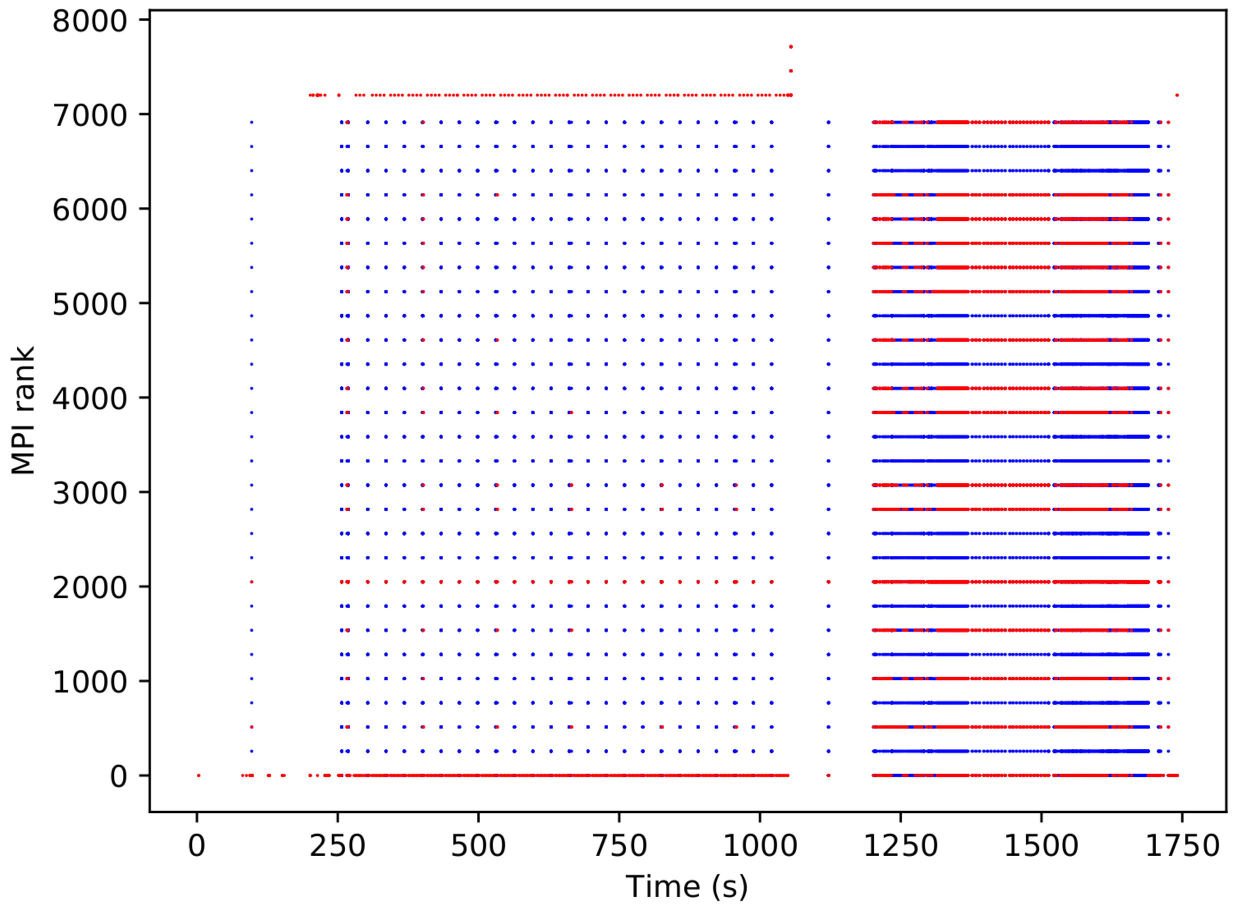


```
pcarns@cori12:~/working/other/nersc-darshan-seminar-2017/logs> darshan-dxt-parser pcarns_ior_id
5599892_6-29-65443-4368632872761953932_1.darshan
```



```
# ***************************************************
# DXT_POSIX module data
# ***************************************************

# DXT, file_id: 11542722479531699073, file_name: /global/cscratch1/sd/pcarns/ior/ior.dat
# DXT, rank: 0, hostname: nid00511
# DXT, write_count: 16, read_count: 16
# DXT, mnt_pt: /global/cscratch1, fs_type: lustre
# DXT, Lustre stripe_size: 1048576, Lustre stripe_count: 24
# DXT, Lustre OST obdidx: 49 185 115 7 135 3 57 95 43 27 191 1 163 51 15 153 187 55 151 239 79
25 137 47
# Module    Rank  Wt/Rd  Segment          Offset       Length    Start(s)      End(s)  [OST]
 X_POSIX       0  write        0               0      1048576      0.7895      0.8267  [ 49]
 X_POSIX       0  write        1         1048576      1048576      0.8267      0.9843  [185]
 X_POSIX       0  write        2         2097152      1048576      0.9843      1.0189  [115]
 X_POSIX       0  write        3         3145728      1048576      1.0189      1.0250  [  7]
```

Detailed trace data



- You can also plot trace data using "dxt_analyzer.py".
- Example on the left:
  - Looks similar to the timespan plots that you already get in the normal Darshan summary.
  - But it plots every individual operation precisely, rather than just showing ranges of times that each process was performing I/O.
  - Closer inspection of the log can identify exactly when and where problematic access patterns were issued.

Darshan: a recap

- These slides covered some basic usage and tips.
- Refer to facility documentation or these slides when you need to.
- Key takeaways:
  - Tools are available to help you understand how your application accesses data.
  - The simplest starting point is Darshan.
  - It’s likely already instrumenting your application, or can quickly be made to do so.
  - Refer to documentation and site support for help interpreting.
  - You will probably start with a pdf generated by darshan-job-summary.pl.
- We’ll see some other use cases and examples this afternoon.

Darshan hands on exercises

- The hands on exercises include 3 Darshan examples that you can try tonight or as time permits during the day:
  - **helloworld**: a simple application that you can run to test out the Darshan toolchain.
  - **warpdrive** and **fidgetspinner**: applications with A and B versions that you can compare to spot the performance differences (and their cause).

The warpdrive and fidgetspinner examples will be easier to understand after seeing the MPI-IO presentation later this morning.

Check with the instructors to share what you find!

Thank you!